Construction of Asynchronous Communicating Systems: Weak Termination Guaranteed!

Kees M. van Hee, Natalia Sidorova, and Jan Martijn van der Werf

Department of Mathematics and Computer Science,
Technische Universiteit Eindhoven,
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
{k.m.v.hee,n.sidorova,j.m.e.m.v.d.werf}@tue.nl

Abstract. Correctness of asynchronously communicating systems (ACS) is known to be a hard problem, which became even more actual after the introduction of Service Oriented Architectures and Service Oriented Computing. In this paper, we focus on one particular correctness property, namely weak termination: at any moment of the system execution, at least one option to terminate should be available. We present a compositional method for constructing an ACS that guarantees weak termination. The method allows for refinement of single components, refinement of compositions of components and the creation of new components in the system. For two important classes of ACS, weak termination follows directly from their structure. These classes focus on the concurrency over components and on the implementation of protocols and communicating choices.

1 Introduction

Verification of asynchronously communicating systems is known to be a hard problem. In past years, modeling and verification mainly focussed on business processes. With the introduction of paradigms like Service Oriented Architectures (SOA) [1, 4, 7], the focus shifts more and more to the modeling and verification of inter organizational processes. Different organizations form a virtual organization to deliver a certain service to other organizations. Languages like the Business Process Execution Language (BPEL) [5] and the current draft of the Business Process Modeling Notation 2.0 (BPMN2) [13] are introduced to model the interaction between processes.

In the paradigm of SOA, components deliver *services* to other components. Communication between components is via message sending, and therefore asynchronous. One of the main aspects of SOA is *dynamic coupling* of components: to perform a service, a component may need services of other components, which might be chosen during runtime. This way, during runtime a tree of service instances is formed. We call this a *service tree*. Due to the dynamic coupling, but also for privacy reasons, components only know their direct neighbors. Hence, the whole service tree is not known to any component in the tree, a component only knows to whom it is connected to. This dynamic nature makes the verification of a service tree very hard, or even infeasible. In this paper we focus on one correctness property, called weak termination, that can be checked compositionally, by checking pairwise compositions of components.

Weak termination means that at any moment of the system execution, at least one option to terminate exists. Note that weak termination does not require that the system always eventually terminates, we only guarantee that the option to terminate always remains open. Weak termination guarantees that a system cannot deadlock nor can it be trapped in an infinite loop: in each infinite loop there is always an option to exit the loop. As the communication between components is asynchronous, Petri nets [14] are a natural choice for modeling the components and their interactions. In [2], the authors provide a method to verify weak termination of service trees compositionally. Each component should be weakly terminating, and each connected pair of components should satisfy a certain condition. This way, a given service tree can be checked by only verifying each component and each connected pair of components.

In this paper, we show a different approach. Instead of checking an existing service tree, we present a *construction methodology* for a class of service trees which are always weakly terminating. The methodology consists of three rules. The first rule allows for the *refinement* of existing components by refining a single place by a new component. The second rule enables the enrichment of the interaction between components by the *replacement* of pairs of places by coupled components that have the weak termination property. The last rule allows for the *creation* of new components as an offspring of an existing component. On top of this rules, we may apply classical refinement rules, e.g., as defined by Murata [12], Berthelot [8], and the workflow refinement rule [10], to refine the internal structure of the component.

For two base classes of coupled components, weak termination can be decided based on their structure. Both classes occur frequently in the design of components, and are based on two subclasses of Petri nets: marked graphs and state machines. The first class focusses on concurrency over components, whereas the latter focusses on communicating choices over components and the design of interaction protocols.

This paper is structured as follows. In Section 2, we introduce some basic concepts and notations. The component framework is explained in Section 3. We present the construction methodology in Section 4, and in Section 5 we introduce the two base classes of coupled components that are weakly terminating by their structure. Finally, we conclude our paper in Section 6.

2 Preliminaries

Let S be a set. The powerset of S is defined as $\mathcal{P}(S) = \{S' \mid S' \subseteq S\}$. With $|S|$ we denote the number of elements in S. The empty set, i.e., the set without any elements is denoted by $\emptyset$. Two sets S and T are disjoint if $S \cap T = \emptyset$. A partition $P \subseteq \mathcal{P}(S)$ is a set such that $\bigcap_{A \in P} A = S$ and $A \cap A' \neq \emptyset \implies A = A'$ for all $A, A' \in P$. We denote the set of all natural numbers as $\mathbb{N} = \{0, 1, 2, \ldots\}$.

A *sequence* σ of length $n \in I\!N$ over S is a function $\sigma : \{1, \ldots, n\} \rightarrow S$. We denote the length of a sequence by $|\sigma| = n$. We denote a sequence of length n by $\sigma = \langle a_1, \ldots, a_n \rangle$ for some $a_1, \ldots, a_n \in S$. If $|\sigma| = 0$, it is the *empty sequence* ϵ. The set of all finite sequences over S is denoted by S^*. *Concatenation* of two firing sequences $\sigma, \upsilon \in S^*$ is a function $\sigma; \upsilon : \{1, \ldots, |\sigma| + |\upsilon|\}$ defined by $(\sigma; \upsilon)(i) = \sigma(i)$ for $1 \leq i \leq |\sigma|$ and $(\sigma; \upsilon)(i) = \upsilon(i - |\sigma|)$ for $|\sigma| < i \leq |\sigma| + |\upsilon|$. The Parikh vector of a sequence σ, denoted by $\overrightarrow{\sigma}$, is a bag representing the number of occurrences of each element in σ. A *bag* m (*multiset*) over S is a function $m : S \rightarrow I\!N$. For $s \in S$, $m(s)$ denotes the number of occurrences of s in m. We denote a bag by square brackets. E.g., in a bag $[a, b^2, c]$, element a occurs once, element b twice, and element c once. All other elements have a multiplicity of 0. We write $I\!N^S$ for the set of all bags over S. The empty bag, i.e., for all elements the multiplicity is 0, is denoted by $\emptyset$. We use $+$ and $-$ for the sum and difference of two bags, and $=$, $<$, $>$, $\leq$, $\geq$ for the comparison of two bags, which are defined in a standard way. Sets can be seen as a special kind of bag were all elements occur only once.

A Petri net N is a tuple (P, T, F) where P is the set of *places*, T is the set of *transitions*, P and T are disjoint, and $F \subseteq (P \times T) \cup (T \times P)$ is the set of *arcs*. An element of $P \cup T$ is called a *node*. Graphically, we denote places by circles, transitions by squares, and arcs as arrows between places and transitions. The preset $^\bullet n$ of a node $n \in P \cup T$ is defined as $^\bullet n = \{n' \in P \cup T \mid (n', n) \in F\}$. Its postset $n^\bullet$ is defined as $n^\bullet = \{n' \in P \cup T \mid (n, n') \in F\}$. The *state* of a Petri net, called a *marking* is a bag over the places P of N. A marking is graphically represented by placing *tokens* in each place. A marked Petri net is a pair (N, m_0), where N is a Petri net and m_0 a marking of N. A transition $t \in T$ is *enabled* in (N, m_0), denoted by $(N : m_0 \xrightarrow{t})$ if $^\bullet t \leq m_0$. An enabled transition in (N, m_0) can *fire* resulting in a new marking $m' = m_0 - {}^\bullet t + t^\bullet$, denoted by $(N : m_0 \xrightarrow{t} m')$. We lift the notation of transition firing and enabledness to sequences in a standard way. A sequence $\sigma \in T^*$ of length $n \in I\!N$ is a *firing sequence* of (N, m_0) if there exist markings $m_{i-1}, m_i \in I\!N^P$ such that $(N : m_{i-1} \xrightarrow{\sigma(i)} m_i)$ for all $1 \leq i \leq n$, and is denoted by $(N : m_0 \xrightarrow{\sigma} m_n)$. The set of all reachable markings of (N, m_0) is defined as $\mathcal{R}(N, m_0) = \{m \mid \exists \sigma \in T^* : (N : m_0 \xrightarrow{\sigma} m)\}$. The set of all possible firing sequences from m_0 to m is denoted by $\mathcal{L}(N, m_0, m) = \{\sigma \in T^* \mid (N : m_0 \xrightarrow{\sigma} m)\}$. A place is k*-bounded* in (N, m_0) for some $k \in I\!N$ if $m(p) \leq k$ for all $m \in \mathcal{R}(N, m_0)$. A marked Petri net is k*-bounded* if all places are k-bounded. A place or marked Petri net is *safe* if it is 1-bounded. A marking m of N is a *deadlock* if there are no transitions enabled in (N, m). It is a *home marking* of (N, m_0) if $m \in \mathcal{R}(N, m')$ for all $m' \in \mathcal{R}(N, m_0)$.

Two Petri nets $N_1 = (P_1, T_1, F_1)$ and $N_2 = (P_2, T_2, F_2)$ are *isomorphic* with respect to some function $\rho : P_1 \cup T_1 \rightarrow P_2 \cup T_2$ if ρ is bijective, $\rho(p) \in P_2$ for all $p \in P_1$, $\rho(t) \in T_2$ for all $t \in T_1$ and $(p, t)\imath F_1$ if and only if $(\rho(p), \rho(t)) \in F_2$.

If for a Petri net $N = (P, T, F)$ we have $|^\bullet t| \leq 1$ and $|t^\bullet| \leq 1$ for all $t \in T$, the Petri net is an *S-net*, also called a *state machine*. If $|^\bullet p| \leq 1$ and $|p^\bullet| \leq 1$ for all

$p \in P$, it is a *T-net*, also called a *marked graph.* If a net is a state machine, we graphically omit the transitions between places.

A special class of Petri nets are *workflow nets.* A workflow net is a Petri net $N = (P, T, F)$ such that there exist exactly one place $i \in P$ with ${}^\bullet i = \emptyset$, called the *initial place*, one place $f \in P$ with $f^\bullet = \emptyset$, called the *final place*, and all nodes $n \in P \cup T$ are on a path from i to f. A workflow net is *sound* if $[f]$ is a home marking of $(N, [i])$ and for all transitions $t \in T : \exists m \in \mathcal{R}(N, [i]) : {}^\bullet t \leq m$. A workflow net N is *generalized sound* if $[f^k]$ is a home marking of $(N, [i^k])$ for all $k \in I\!N$.

3 Asynchronous Communicating Systems

A system consists of components that communicate asynchronously with each other via interfaces, and to each interface at most one component is connected. In this approach, we model a component by a Petri net [14]. As communication is asynchronous, we model the communication via special places, called *interface places* (cf. [11]). An interface place is either an *input place*, i.e., the component receives a message via this place, or an *output place*, i.e., the component sends a message via this place. As a component needs to communicate with other components, the interface places are partitioned into *ports.* Transitions have a sign with respect to a port. For each port, a transition either sends messages (sign !), it receives messages (sign ?), or it is silent (sign τ). Consider the component shown in Figure 1. In this example, component N has three ports, G, H, and J. Port G consists of two input places b and e, and three output places a, c and d. Transition t has sign ! with respect to port G and sign τ with respect to port H.

The internal places together with the transitions form the inner structure of the component, which we call the *skeleton.* The input and output places determine the interfaces to the exterior. A component has one initial and one final marking, in which only internal places are marked, i.e., in the initial and final markings no interface places can be marked. The final marking does not need to be a deadlock.

Definition 1 (Component, skeleton, sign). *A* component *is an 8-tuple* $(P, I, O, T, F, \mathcal{G}, i, f)$ *where* $((P \cup I \cup O, T, F), i)$ *is a marked Petri net; P is a set of* internal places*; I is a set of* input places*, and* ${}^\bullet I = \emptyset$*; O is a set of* output places*, and* $O^\bullet = \emptyset$*; P, I, O are pairwise disjoint; $\mathcal{G} \subseteq \mathcal{P}(I \cup O)$ is a partition of the interface places, called the* ports*. A transition either sends to or receives from a port, i.e.,* ${}^\bullet G \cap G^\bullet = \emptyset$ *for all* $G \in \mathcal{G}$*.* $i \in I\!N^P$ *is the* initial marking*; and* $f \in I\!N^P$ *is the* final marking*. We call the set $I \cup O$ the* interface places *of the component. Two components N and M are called* disjoint *if P_N, P_M, I_N, I_M, O_N, O_M, T_N and T_M are pairwise disjoint.*

The skeleton *of an OPN N is defined as the Petri net $\mathcal{S}(N) = (P_N, T_N, F)$ with $F = F_N \cap ((P_N \times T_N) \cup (T_N \times P_N))$. The* sign *of a transition with respect to a port $G \in \mathcal{G}$ is a function $\lambda_G : T \to \{!, ?, \tau\}$ defined by $\lambda_G(t) =!$ if $t^\bullet \cap G \neq \emptyset$, $\lambda_G(t) =?$ if ${}^\bullet t \cap G \neq \emptyset$, and $\lambda_G(t) = \tau$ otherwise, for all $t \in T_N$.*

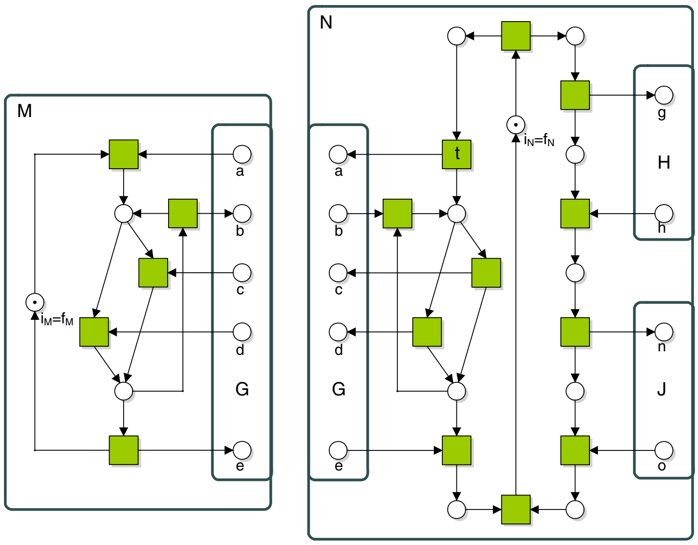


Fig. 1. A component with three ports G, H and J

Even if a component is not connected to any other component, it should work correctly, i.e., it should always be possible to reach the final marking of the component. This property is called *weak termination.*

Definition 2 (Weak termination of a component). *A component N is* weakly terminating *if f_N is a home marking of $(\mathcal{S}(N), i_N)$.*

We do not require the final marking to be a deadlock. Instead, the final marking can be a state from which it is always possible to return to, called an *idle state* in which the component is in rest. For example, if the initial marking and final marking are identical, this is the case.

Components communicate via their ports. Communication is only possible if the input places of the one component are the output places of the other component and vice versa. In our approach we compose components by fusing interface places with the same name.

Definition 3 (Composition). *The components A and B are* composable *if there exists a port $G \in \mathcal{G}_A \cap \mathcal{G}_B$ such that $(I_A \cap O_B) \cup (O_B \cap I_B) = G$ and $P_A \cap P_B = T_A \cap T_B = I_A \cap I_B = O_A \cap O_B = \emptyset$. If A and B are composable, their* composition *is a component $A \oplus_G B = (P, I, O, T, F, \mathcal{G}, i, f)$ defined by: $P = P_A \cup P_B \cup G$, $T = T_A \cup T_B$, $F = F_A \cup F_B$; $I = (I_A \setminus G) \cup (I_B \setminus G)$; $O = (O_A \setminus G) \cup (O_B \setminus G)$; $\mathcal{G} = (\mathcal{G}_A \cup \mathcal{G}_B) \setminus \{G\}$; $i = i_A + i_B$; and $f = f_A + f_B$. If there exists a unique port $G \in \mathcal{G}_A \cap \mathcal{G}_B$ such that $A \oplus_G B$, we write $A \oplus B$.*

Consider again the example of Figure 1. Components N and M share port G, and all input places of port G in N are output places of port G in M and vice versa. Hence, we can compose the two components via port G. In the resulting net, the interface places of port G become internal places of the composition $N \oplus_G M$.

4 Construction Rules

To guarantee weak termination of a system consisting of asynchronous communicating components is in general very hard due to high degree of concurrency of the components. In [2], a sufficient condition has been presented to pairwise verify weak termination for a tree-structured composition of components. In this section we present an approach that guarantees this condition by construction. The approach consists of three rules to refine a system. The first rule is refinement within a single component. In the second rule, two so called "synchronized places" can be refined by a composition of two components. The last rule involves the creation of a new coupled component in a system.

4.1 Refinement within Components

For Petri nets there already exist many refinement rules, like the rules of Murata [12] and Berthelot [8]. These rules guarantee weak termination: applying them on a weakly terminating component results again in a weakly terminating component. However, these rules are all applied on internal parts of the component and do not extend the ports of a component. In [10], the authors show that a place in a workflow net may be refined by a generalized sound workflow net, while preserving the soundness condition. We redefine this refinement operation on components and refine an internal safe place p of a component N by a workflow component M. A workflow component is a component which has a workflow net as skeleton, the only marked place in the initial marking is the initial place of the skeleton, and the only marked place in the final marking is the final place of the skeleton.

Definition 4 (Workflow component). *A component N is a* workflow component *if $\mathcal{S}(N)$ is a workflow net with initial place i and final place f, and $i_N = [i]$ and $f_N = [f]$.*

When we refine a place p in a component N by a new workflow component M, all transitions in the preset of p are connected to the initial place of M, and all transitions in the postset of p are connected to the final place of M. Place p is then removed from N. The ports of M are added to the ports of N. Consider the example of Figure 2. In this example, place p of component N is refined by component M. In the refined net N', the port G of M is added to the already existing ports of N.

Definition 5 (Place refinement). *Let N be a component and M be a workflow component, such that N and M are disjoint. Let $p \in P_N$ be a place that is safe in $(\mathcal{S}(N), i_N)$. The refined component $N \odot_p M = (P, I, O, T, \mathcal{G}, F, i, f)$ is defined as:*

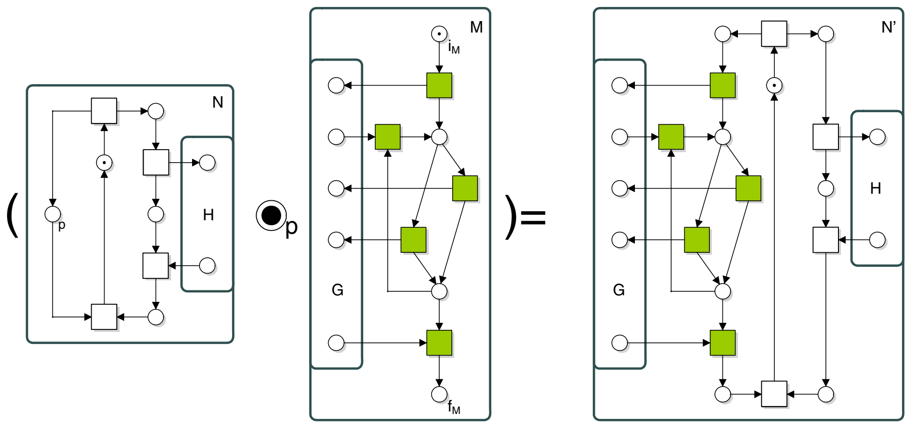


Fig. 2. Refinement of place p in N by component M

$P = (P_N \setminus \{p\}) \cup P_M$; $I = I_N \cup I_M$; $O = O_N \cup O_M$; $T = T_N \cup T_M$; $\mathcal{G} = \mathcal{G}_N \cup \mathcal{G}_M$ $F = F_N \setminus ((^\bullet p \times \{p\}) \cup (\{p\} \times p^\bullet)) \cup F_M \cup (^\bullet p \times \{i_M\}) \cup (\{f_M\} \times p^\bullet)$; $i(s) = i_N(s)$ *and* $f(s) = f_N(s)$ *for all* $s \in P_A \setminus \{p\}$, $i(i_M) = i_N(p)$ *and* $f(f_M) = f_N(p)$.

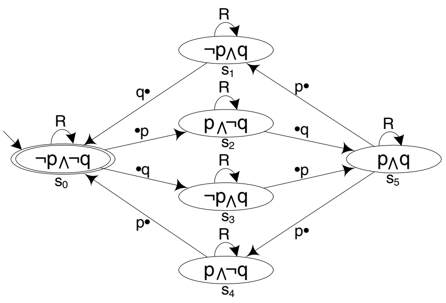
This definition of place refinement propagates the ports of the refining component to the original component. At a first glance, this definition seems to contradict the paradigm of information hiding. However, the definition allows for the refinement of a component by a composed component, as long as this composition remains a workflow component. This way, the ports remain invisible to the environment of the original component.

Since we only allow to refine safe places, we do not need to require the component with which the place will be refined to be generalized sound as in [10], but weak termination as defined in the previous section is sufficient.

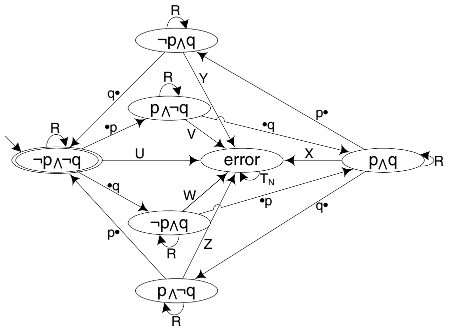
Theorem 6 (Refinement of safe places preserves weak termination). *Let* N *be a component and let* M *be a weakly terminating workflow component. Let* $p \in P_N$ *be a safe place in* $(\mathcal{S}(N), i_N)$. *If both* N *and* M *are weakly terminating, then* $N \odot_p M$ *is weakly terminating.*

4.2 Refinement over Components

In system construction, component interaction is often established in several cycles. In each cycle the interaction is refined, until the desired communication protocol is designed. The first rule, i.e., refinement of a single place in a component, does not suffice, as it creates a new port. Thus, it cannot create a more elaborate interaction protocol between components, it can only create new interactions. Moreover, we want to refine the scheme of interacting components. For example, a simple request-response pattern could be refined in a more elaborated negotiation pattern. Therefore, we introduce refinement of special pairs



(a) $\mathrm{Sync}_{p,q}(N)$



(b) $\overline{\mathrm{Sync}}_{p,q}(N)$

Fig. 3. Desired behavior for synchronized places p and q in component N

of places in a composition. Refinement of a single place in a weakly terminating component by a weakly terminating subcomponent results again in a weakly terminating component. Refinement of two places by a composition of two components is in general not weakly terminating. As we refine a pair of places by a composition, we need to be sure that there exists markings in which both places are marked. If such a marking cannot be reached, the interaction we refine the components with cannot be executed properly, and thus the refined composition is not weakly terminating. An intuitive approach would be to apply this refinement only to "synchronizable" places, i.e., two places such that whenever one is marked before the other, it is always possible to keep the place marked until the other is marked as well.

Consider two places p and q that satisfy this intuitive requirement, i.e, if p becomes marked, then it is always possible to keep it marked until q is marked, or vice versa. To describe the desired behavior on places p and q, we use a state machine as shown in Figure 3(a). The initial marking s_0 is also the final marking. In the annotation of places, p means that p contains a token, and $\neg p$ means that p does not contain tokens (similarly for q). The arcs are annotated by sets of transitions: for each element in the set on an arc from s to s' there is a transition with preset $\{s\}$ and postset $\{s'\}$. The set R is defined as $R = T_N \backslash ({}^\bullet p \cup p^\bullet \cup {}^\bullet q \cup q^\bullet)$. Initially, both places p and q are unmarked. If a transition in the preset of p or q fires, we reach a state in which either p or q is marked. Then, if p is marked only transitions in R or ${}^\bullet q$ are able to fire, or, if q is marked, only transition in R or ${}^\bullet p$ are able to fire. If such a transition fires, we reach a marking in which both places p and q are marked. From this state, first both places need to become unmarked, before they can become marked again.

If any accepting firing sequence, i.e., a firing sequence from initial marking to the final marking, in the skeleton of the component is also an accepting firing sequence of $\mathrm{Sync}_{p,q}(N)$, a pair of places is synchronized. Consider the example of Figure 4. In this example, there exist accepting firing sequences of $A \oplus_G B$ that are also accepting firing sequences of $\mathrm{Sync}_{p,q}(A \oplus_G B)$. Consider the accepting firing sequence $\langle t, u, w, v, x, y, z \rangle$. Then this is not an accepting firing sequence of $\text{Sync}_{p,q}$. However, since transition v does not depend on the input of transition w, we can *swap* these transitions. This way, we can *shuffle* the firing sequence σ to the new firing sequence $\langle t, u, v, w, x, y, z \rangle$. This firing sequence is an accepting firing sequence for both $A \oplus_G B$ and $\text{Sync}_{p,q}$. Hence, if for any accepting firing sequence of $A \oplus_G B$ there exists such a shuffled firing sequence that is an accepting firing sequence of both $A \oplus_G B$ and $\text{Sync}_{p,q}$, places p and q are synchronized. If there would exist an accepting firing sequence that cannot be reshuffled into an accepting firing sequence of both, the interaction we refine with cannot be initiated properly, and thus the refined composition is not weakly terminating anymore.

Definition 7 (Synchronized places). *Let A and B be two component composable with respect to some port $G \in \mathcal{G}_A \cap \mathcal{G}_B$. Two places $p \in P_A$ and $q \in P_B$ are* synchronized*, denoted by $p \rightleftharpoons_N q$ if and only if*

$$\forall \sigma \in \mathcal{L}(N, i, f) : \sigma \in \mathcal{L}(\mathit{Sync}_{p,q}(N))$$

A direct consequence of the definition of $p \rightleftharpoons_N q$ is that the synchronized places p and q are safe in the skeleton of N.

Lemma 8. *Let N be a component, and $p, q \in P_N$ such that $p \rightleftharpoons_N q$. Then p and q are safe in $(\mathcal{S}(N), i_N)$.*

Checking whether two places are synchronized is decidable. The state machine $\overline{\text{Sync}}_{p,q}(N)$ extends $\text{Sync}_{p,q}(N)$ by adding an extra state annotated with *error*. Let $U = p^\bullet \cup q^\bullet$, $V = U \cup {}^\bullet p$, $W = U \cup {}^\bullet q$, $X = {}^\bullet q \cup {}^\bullet p$, $Y = X \cup p^\bullet$, and $Z = X \cup q^\bullet$. Connect the states as shown in Figure 3(b). Then, state *error* is a live lock, which can only be reached if places p and q are not synchronizable. Hence, in the synchronous product of component N and $\overline{\text{Sync}}_{p,q}(N)$, no marking should be reachable in which place *error* is marked, which is a classical coverability problem that is decidable for Petri nets.

Given a composition $N = A \oplus_G B$, the refinement of two synchronized places by a composition of two workflow components $C \oplus_H D$ results in a new composition, where in component A place p is refined by C, and in component B place q is refined by D. The interface places of ports G and H become internal places.

Definition 9 (Refinement of synchronized places). *Let A and B be two components that are composable with respect to port $G \in \mathcal{G}_A \cap \mathcal{G}_B$ and let C and D be two workflow components that are composable with respect to port $H \in \mathcal{G}_C \cap \mathcal{G}_D$ and $A \oplus_G B$ and $C \oplus_H D$ are disjoint. Let $p \in P_A$ and $q \in P_B$ such that $p \rightleftharpoons_{A \oplus_G B} q$. The refined component $(A \oplus_G B)_p \odot_q (C \oplus_H D) = (P, I, O, T, F, \mathcal{G}, i, f)$ is defined by: $P = (P_{A \oplus_G B} \cup P_{C \oplus_H D}) \setminus \{p, q\}$; $T = T_{A \oplus_G B} \cup T_{C \oplus_H D}$; $I = I_{A \oplus_G B} \cup I_{C \oplus_H D}$; $O = O_{A \oplus_G B} \cup O_{C \oplus_H D}$; $F = (P_{A \oplus_G B} \cup P_{C \oplus_H D} \setminus (({}^\bullet p \times \{p\}) \cup (\{p\} \times p^\bullet) \cup ({}^\bullet q \times \{q\}) \cup (\{q\} \times q^\bullet))) \cup ({}^\bullet p \times \{i_C\}) \cup (\{f_C\} \times p^\bullet) \cup ({}^\bullet q \times \{i_D\}) \cup (\{f_D\} \times p^\bullet)$; $i = i_{A \oplus_G B}$; and $f = f_{A \oplus_G B}$.*

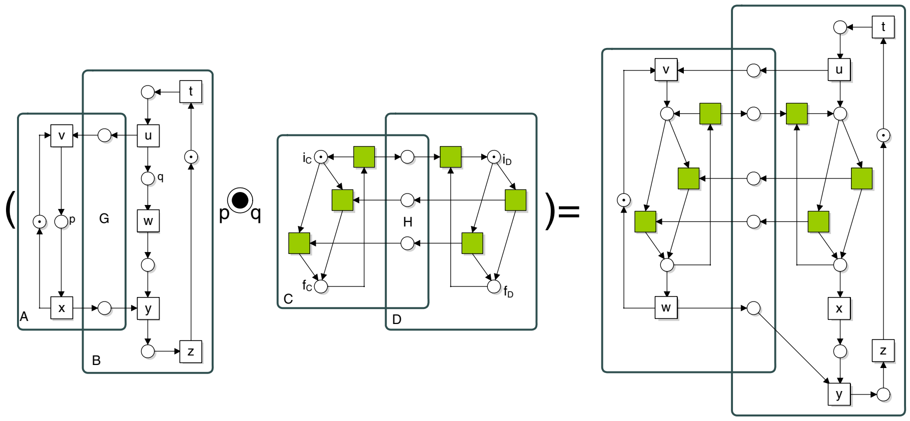


Fig. 4. The refinement $(A \oplus_G B)_p \odot_q (C \oplus_H D)$

Consider the example of Figure 4. In this example, place p of component A and place q of component B are synchronized in $A \oplus_G B$. Components C and D are workflow components. Both compositions $A \oplus_G B$ and $C \oplus_H D$ are weakly terminating. The refined component $(A \oplus_G B)_p \odot_q (C \oplus_H D)$ is also weakly terminating.

Theorem 10 (Weak termination for refinement of synchronized places). *Let A and B be two components that are composable with respect to port $G \in \mathcal{G}_A \cap \mathcal{G}_B$ and let C and D be two workflow components that are composable with respect to port $H \in \mathcal{G}_C \cap \mathcal{G}_D$ and $A \oplus_G B$ and $C \oplus_H D$ are disjoint. Let $p \in P_A$ and $q \in P_B$ such that $p \rightleftharpoons_{A \oplus_G B} q$. If $A \oplus_G B$ and $C \oplus_H D$ are weakly terminating, then the refined component $(A \oplus_G B)_p \odot_q (C \oplus_H D)$ is also weakly terminating.*

4.3 Creating New Components

The first two rules only allow to extend existing components. With the third rule, it is possible to connect new components in a system such that the system remains weakly terminating. The rule is based on the principle of outsourcing. Consider Figure 5. For example, if place p has the meaning that when a token resides in it, "an item is produced", and the decision is taken to outsource the production activity, we can add two transitions: a "start producing item" and a "finish producing item". Then the start transition initiates the component producing the item, and the finish transition fires if the item is produced. Creating a new port for these transitions allows the connection of a new component to the existing system. To realize the intended refinement, a place p in a component N is refined by the component M_1 – a component that sends a message over a new port G and then waits in place x until it receives a message on port G.

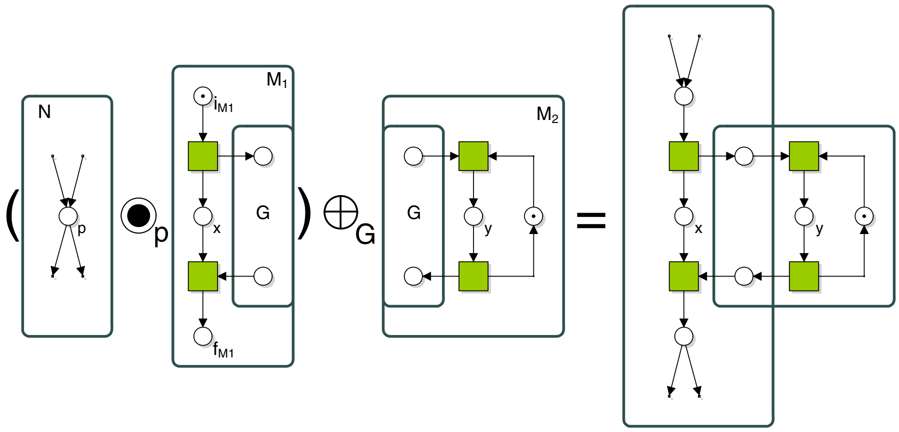


Fig. 5. Extending the composition with a new coupled component

Component M_1 is composed to a component M_2 that waits in an idle state until it receives a message on port G, thus marking place y. M_2 then sends a message, and returns to its idle state. In BPEL, this corresponds to an invoke activity, where a message is sent to execute an operation, and the activity waits for the result of the operation. There are many variants of this rule possible.

If in a weakly terminating component the place we extend is safe, the newly created system is again weakly terminating. Furthermore, the newly added places x and y are synchronized by construction. Hence, we can apply the second rule of the approach on these places to refine the interaction between $N \odot_p M_1$ and M_2.

Theorem 11. *Let N be a component, $M_1 \oplus M_2$ the request-response net as depicted in Figure 5, such that N and M_1 are disjoint and N and M_2 are disjoint. Define $N' = (N \odot_p M_1) \oplus M_2$. Then N' is weakly terminating and $x \rightleftharpoons_{N'} y$.*

5 Basic Classes of Weakly Terminating Compositions

In the previous section, we presented a construction approach to build systems that are weakly terminating by construction. However, for the second rule, i.e., to refine two synchronized places, we need a weakly terminating composition of two workflow components. In this section, we present two basic classes of communicating components for which the communication condition can be decided based on their structure. The first class is based on marked graphs, and introduces concurrency within components and concurrent communication in compositions. The second class is based on state machines, to build more complex communication interactions and protocols.

5.1 Acyclic Marked Graph Components

A special subclass of Petri nets are marked graphs. In a *marked graph*, or T-net, all places have a preset and postset of length at most one. We extend this notion to components: a workflow component is a *T-component* if its skeleton is a marked graph, and all interface places are connected to exactly one transition. In [10], the authors show that if a T-workflow, i.e., a workflow that is also a T-net, is acyclic, it is generalized sound. We can use these results to obtain a similar result for T-components: if a T-component is acyclic, it is safe and weakly terminating.

Lemma 12 (Weak termination and safeness and T-components [10]). *Let N be a T-component such that $\mathcal{S}(N)$ is acyclic. Then it is weakly terminating and safe.*

By definition of a T-component, each interface place in a component has at most one transition connected to it. Hence, if two T-components are composable with respect to some port, in their composition these interface places have one transition in their preset and one transition in their postset, and thus the composition is again a T-component. From Lemma 12, we may directly conclude that if the composition of two T-components is acyclic, the composition is weakly terminating and safe.

Theorem 13 (Weak termination and safeness for compositions of T-components). *Let N and M be two acyclic T-components composable with respect to some port $G \in \mathcal{G}_N \cap \mathcal{G}_M$ such that $N \oplus_G M$ is acyclic. Then $N \oplus_G M$ is safe and weakly terminating.*

As a result, we may refine two synchronized places in a composition by an acyclic composition of T-components. The algorithm presented in the previous section can be used to determine the pairs of synchronized places. If places in a T-component are causal independent, i.e., it is possible that both places are marked in a place, then they are synchronized.

5.2 Isomorphic State Machine Components

A second subclass of Petri nets are state machines. A *state machine* is the dual of a marked graph: each transition has a preset and a postset of length at most one. A workflow component is an *S-component* if its skeleton is a state machine, and each interface place is connected to exactly one transition. From [10], it is easy to conclude that S-components are always weak terminating and safe.

Lemma 14 (Weak termination and safety of S-components). *Let N be an S-component. Then it is weakly terminating and safe.*

Although S-components have a simple structure, their composition is not. Composing S-components introduces concurrency; it is very simple to compose two

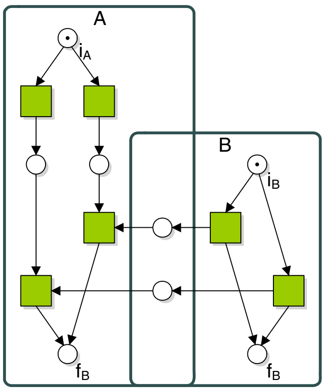


Fig. 6. Classical problem with point where decision is taken. A receiving gives problems.

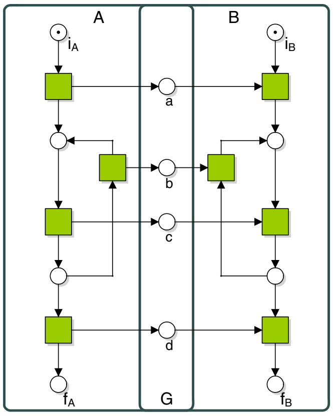


Fig. 7. The direction of communication via b and c matters. If both have the same sign, the composition $A \oplus B$ is not sound.

S-components such that the resulting composition can never reach the final marking, or even has deadlocks. We illustrate this with some examples. In Figure 6 we see a classical example showing that the composition of two sound S-components is not sound anymore. The choice A made is not communicated to component B. Hence, both can make a different choice, thus entering a deadlock. A solution to overcome this problem is to only connect S-components that have isomorphic skeletons, and communicate all choices. However, in Figure 7 we see two isomorphic S-components and although all choices made by A are communicated, we see that $A \oplus B$ is not weakly terminating, since in A the loop may be executed more times than component B executes the loop. Hence, tokens remain in the interface places, and thus the composition is not weakly terminating.

The composition of two T-components resulted in a T-component again. A similar property does not hold for S-components. The communicating transitions violate the state machine property in the composition: in the composition either their preset or their postset contain two places. These examples show that we need strong requirements to allow the composition of two S-components. First, we require the S-components to have isomorphic skeletons, and all transitions communicate, but only to the transition it is isomorphic to. Hence, all transitions either send or receive on the port that is used for the composition. Secondly, we require that if a component makes a choice, i.e., two or more transitions are enabled in a marked place, this choice is communicated, since otherwise the other component does not know in what state the first component is. This means that if two transitions share a place in their preset, they have the same sign. Last, we require that in every loop, there are at least two transitions with a different sign. As all transitions either send or receive, the last requirement implies the existence of both a sending transition and a receiving transition in each loop. If all the requirements hold, we say that the composition agrees on the isomorphism.

Definition 15 (Composition agrees on isomorphism). *Let A and B be two S-components such that their skeletons are isomorphic with respect to ρ. The composition $N = A \oplus_G B$ for some port $G \in \mathcal{G}_A \cap \mathcal{G}_B$* agrees on ρ *if and only if:*

- *for all transitions $t \in T_A$, $t' \in T_B$, there exists a place $s \in G$ such that $\{(t,s),(s,t')\} \subseteq F_N$ or $\{(t',s),(s,t)\} \subseteq F_N$ if and only if $\rho(t) = t'$;*
- *All transitions in the postset of a place have the same sign, i.e. $\forall p \in P_N, t_1, t_2 \in p^\bullet : \lambda_G(t_1) = \lambda_G(t_2)$;*
- *For all markings $m \in \mathcal{R}(\mathcal{S}(A), i_A)$ and firing sequences $\sigma \in T_A^*$ such that $(\mathcal{S}(A) : m \xrightarrow{\sigma} m)$ there are $i, j \in \{0, \ldots |\sigma|\}$ such that $\lambda_G(\sigma(i)) =! \wedge \lambda_G(\sigma(j)) =?$.*

Although isomorphism is a strong requirement, in practice it is often used for protocol design between agents. First a state machine is designed that represents the communication between the two agents. In each state of choice, only one agent can make a choice. Then the state machine is copied for both agents, and the communication between the transitions is added. Definition 15 gives a set of rules for asynchronous communication between these transitions such that the composition is always weakly terminating and safe.

Theorem 16. *Let N and M be two composable S-components with respect to some port $G \in \mathcal{G}_N \cap \mathcal{G}_M$ such that their skeletons are isomorphic with respect to some bijective function ρ. If the composition $N \oplus_G M$ agrees on ρ, $N \oplus_G M$ is weakly terminating and safe.*

If the skeletons of two components B and C are isomorphic, and their composition agrees on this isomorphism, the markings reachable in the composition have a special form: each marking consists of a marked place of B, a marked place of C and some marked interface places. As a consequence of the structure of the composition, it is always possible to mark a place p and its isomorphic place $\rho(p)$, without any interface marked.

In Figure 8 an example system is shown that is constructed using the presented approach. The system was constructed by starting with a single marked place. By the standard refinement rules of Murata [12], the component is refined to a simple marked graph with two places in parallel. Using the third rule, for both places a new component is created. Next, the second rule is applied twice to refine the two synchronized places by a composition of two S-components that agree on the isomorphism of their skeletons, and an acyclic composition of T-components. By construction, the system is weakly terminating.

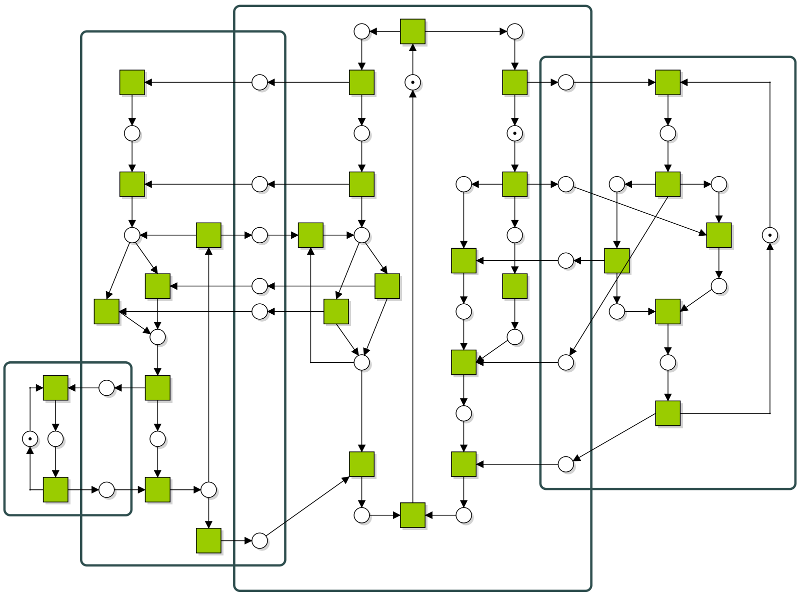


Fig. 8. System constructed using the constuction method

6 Conclusions

In this paper, we presented a construction methodology for asynchronously communicating systems that guarantees weak termination. We model components and their interaction with Petri nets. The construction method consists of three rules. The first rule allows for the refinement of safe places by a weakly terminating workflow component. If the initial component is weakly terminating, the refined component is also weakly terminating. In the second rule, we refine two synchronized places by a weakly terminating composition of two workflow components. A pair of places is synchronized if it is always the case that if one place is marked before the other, it is always possible to keep that place marked until the other is marked as well. If the original composition is weakly terminating, the refined composition is weakly terminating as well. The third rule allows the creation of new coupled components.

We studied two classes of coupled components that are weakly terminating by their structure. The first class is based on marked graphs. If a composition of two workflow T-components is an acyclic marked graph, the composition is weakly terminating. The second class is based on state machines. If the composition of two S-components that are isomorphic on their skeleton agrees on the isomorphism, the composition is weakly terminating.

In [6] the authors give a constructive method preserving the inheritance of behavior, which can be used to guarantee the correctness of interorganizational processes. Other formalisms, like interface automata [3] use synchronous communication, whereas we focus on asynchronous communication, which is essential for our application domain, since the communication in SOA is asynchronous.

In [9], the authors propose to model choreographies using Interaction Petri nets, which is a special class of Petri nets, where transitions are labeled with the source and target component, and the message type being sent. To check whether the composition is functioning correctly, the whole network of components needs to be checked, whereas in our approach this is guaranteed by construction.

To keep our results at a conceptual level, we present our results on Petri net models. Our method can easily instaciated for industrial languages like BPEL, to facilitate the construction of web services in development environments like Oracle BPEL or IBM Websphere.

References

1. van der Aalst, W.M.P., Beisiegel, M., van Hee, K.M., König, D., Stahl, C.: An SOA-Based Architecture Framework. International Journal of Business Process Integration and Management 2(2), 91–101 (2007)
2. van der Aalst, W.M.P., van Hee, K.M., Massuthe, P., Sidorova, N., van der Werf, J.M.E.M.: Compositional service trees. In: Franceschinis, G., Wolf, K. (eds.) ICATPN 2009. LNCS, vol. 5606, pp. 283–302. Springer, Heidelberg (2009)
3. de Alfaro, L., Henzinger, T.A.: Interface automata. SIGSOFT Softw. Eng. Notes 26(5), 109–120 (2001)
4. Alonso, G., Casati, F., Kuno, H., Machiraju, V.: Web Services Concepts, Architectures and Applications. Springer, Heidelberg (2004)
5. Alves, A., Arkin, A., Askary, S., et al.: Web Services Business Process Execution Language Version 2.0 (OASIS Standard). WS-BPEL TC OASIS (2007), `http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html`
6. Basten, T., van der Aalst, W.M.P.: Inheritance of Behavior. Journal of Logic and Algebraic Programming 47(2), 47–145 (2001)
7. Bell, M.: Service-Oriented Modeling (SOA): Service Analysis, Design, and Architecture. Wiley, Chichester (2008)
8. Berthelot, G.: Transformations and Decompositions of Nets. In: Brauer, W., Reisig, W., Rozenberg, G. (eds.) APN 1986. LNCS, vol. 254, pp. 360–376. Springer, Heidelberg (1987)
9. Decker, G., Weske, M.: Local enforceability in interaction petri nets. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 305–319. Springer, Heidelberg (2007)
10. van Hee, K.M., Sidorova, N., Voorhoeve, M.: Soundness and Separability of Workflow Nets in the Stepwise Refinement Approach. In: van der Aalst, W.M.P., Best, E. (eds.) ICATPN 2003. LNCS, vol. 2679, pp. 337–356. Springer, Heidelberg (2003)
11. Kindler, E.: A compositional partial order semantics for petri net components. In: Azéma, P., Balbo, G. (eds.) ICATPN 1997. LNCS, vol. 1248, pp. 235–252. Springer, Heidelberg (1997)
12. Murata, T.: Petri Nets: Properties, Analysis and Applications. Proceedings of the IEEE 77(4), 541–580 (1989)
13. Object Management Group. Business Process Model and Notation (BPMN) Specification 2.0 V0.9 (November 2008)
14. Reisig, W.: Petri Nets: An Introduction. Monographs in Theoretical Computer Science: An EATCS Series, vol. 4. Springer, Berlin (1985)